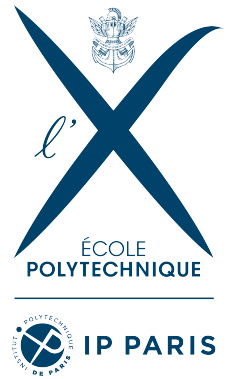
Reinforcement Learning I

INF581 Advanced Machine Learning and Autonomous Agents

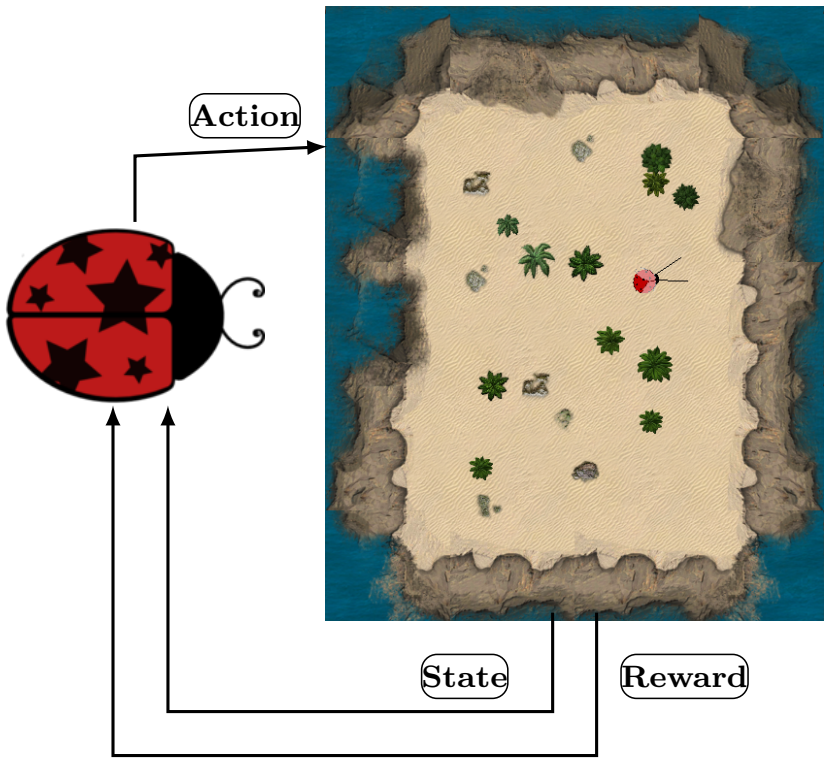
Jesse Read



Last Updated: January 26, 2022

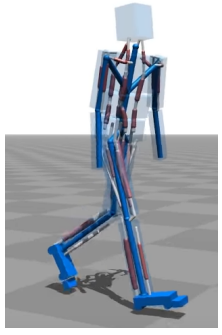
Introduction: Reinforcement Learning

- We are not given a data set; rather an environment
- No training labels; only reward signal
- Sequential decision making (non-i.i.d. data)
- The model is an agent; map state-observations to actions
- Main challenge: Reward signal is delayed, sparse, and/or weak

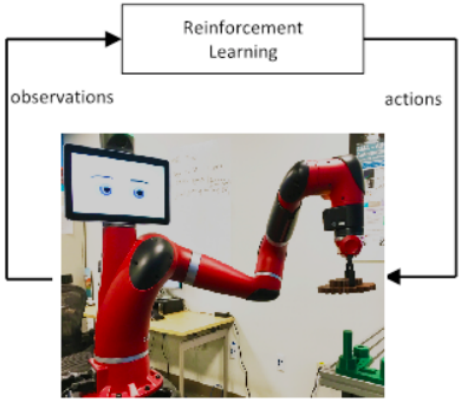


Introduction: Applications and Main Concepts

Applications: Fun and Robots

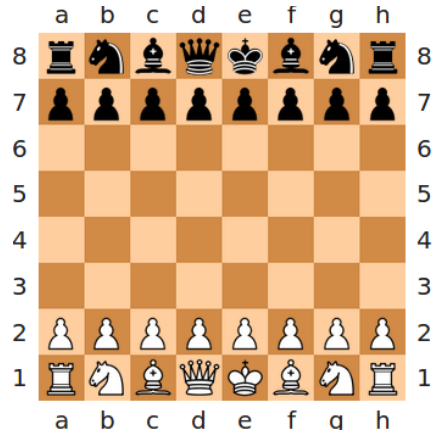


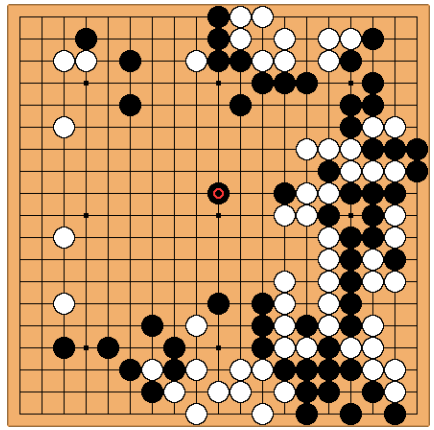
▸ AIGym ▸ Walker1 ▸ Walker2 ▸ Cars

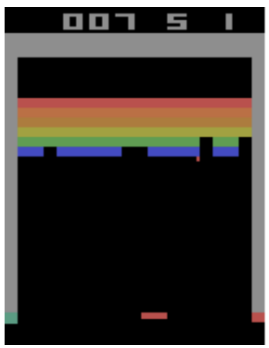


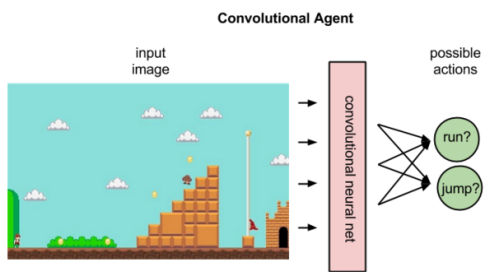
▸ Helicopter ▸ Robot ▸ More Robots

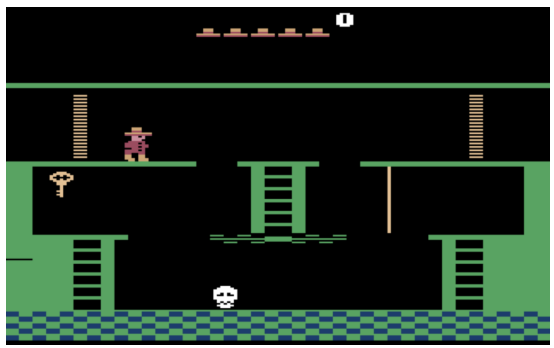
Applications: Games

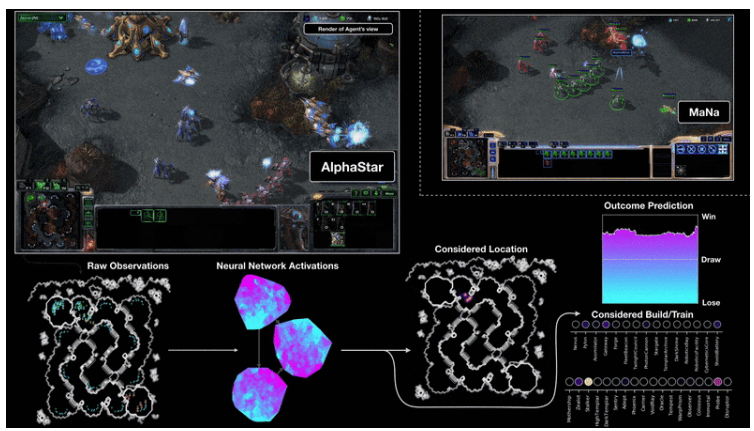








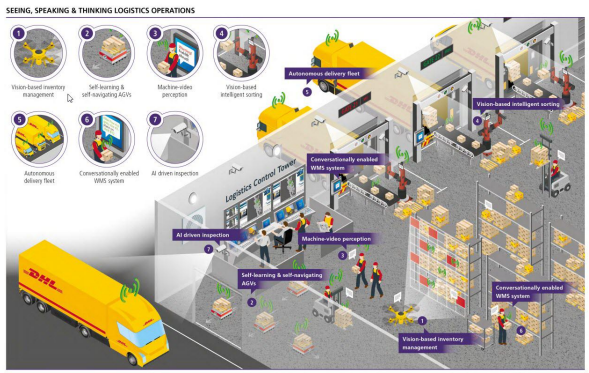


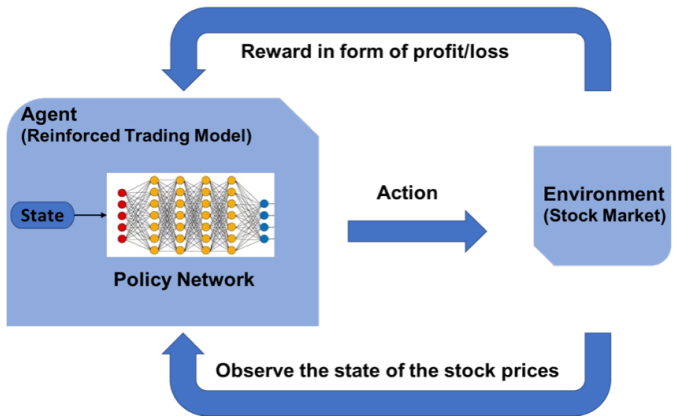


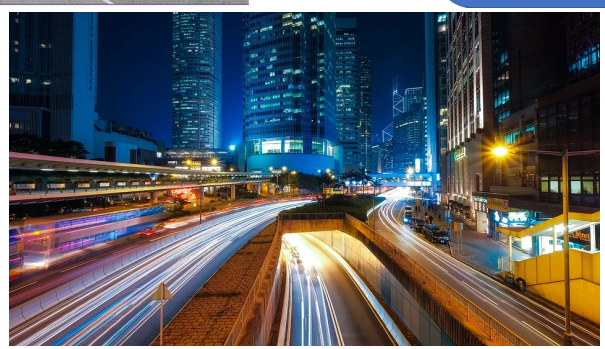
▸ FPS-Quake ▸ Starcraft-AlphaStar

Applications: Finance and Business

- Logistics; supply and demand (products, energy, . . .) ▸ Energy
- Trading/finance/manage investment portfolio ▸ IBM
- In recommendation engines, email advertising
- Marketing and advertising; real-time bidding ▸ Advertising/Bidding ▸ Bidding

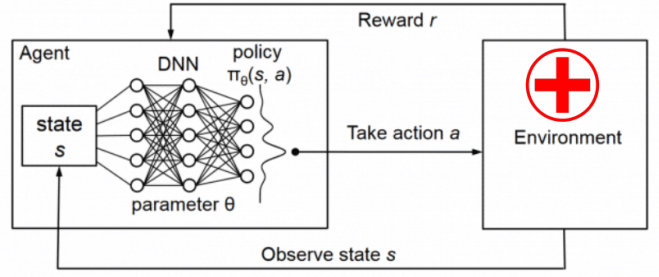






Applications: Education, Healthcare, Agriculture, . . .

- Healthcare: Diagnosis, manage hospital resources, . . . ▸ Health
- Education: Personalised/auto-generated curriculum, . . .
- Farming and Agriculture: Managing resources, . . .
- Sports
- Politics: Obtaining a policy ▸ Politics

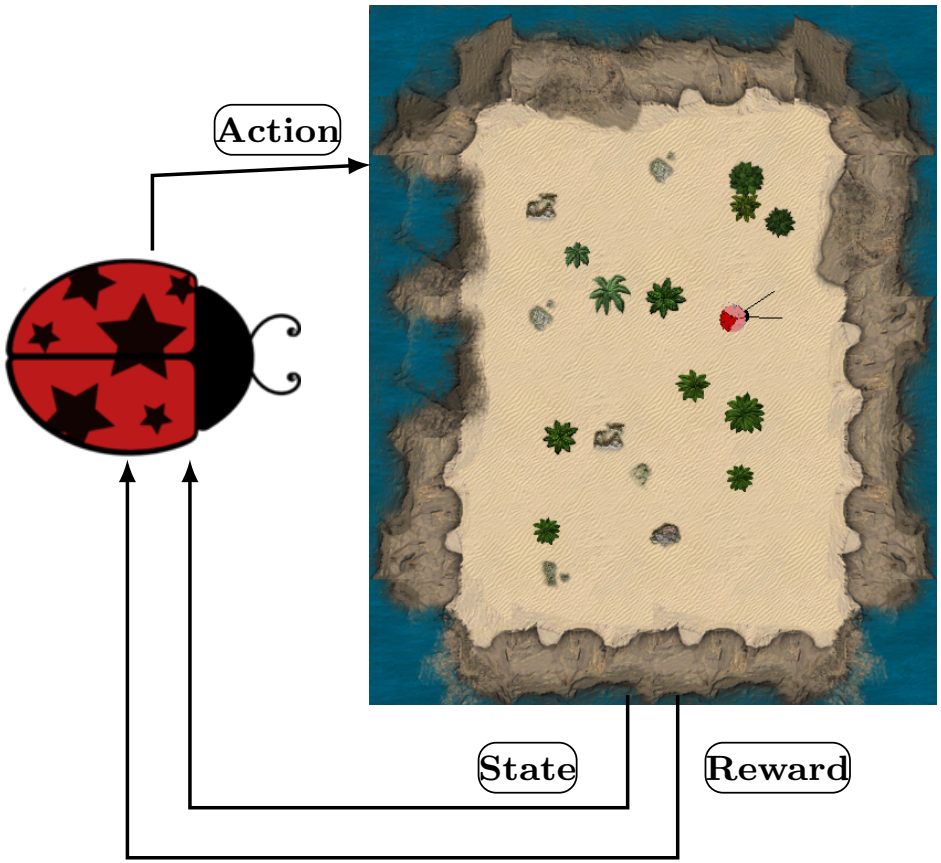


In NLP and Machine Learning

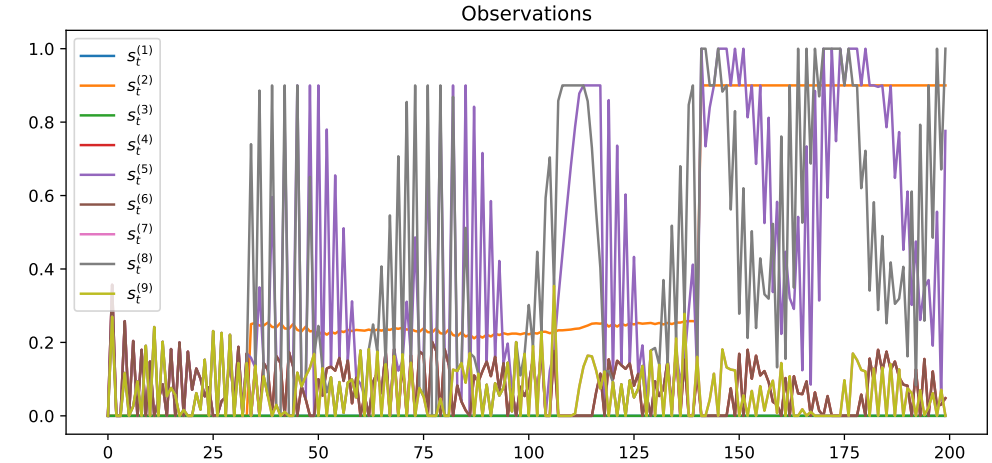
- Auto-tune the parameters of a deep neural network
- In chatbots
- In machine translation

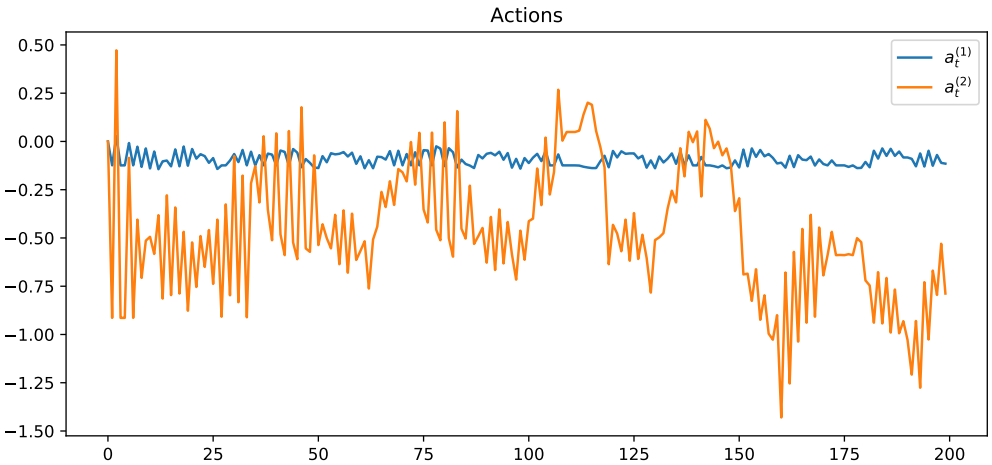
Number of real-world applications of reinforcement learning is increasing rapidly.

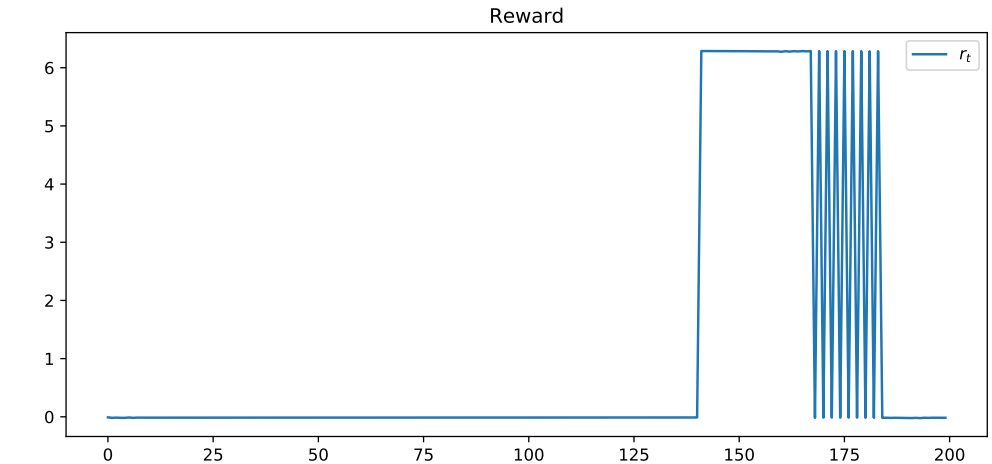
The Agent and the Environment



1. Observe the state of the environment
2. Obtain reward
3. Goal: select action to maximize future rewards.


Observations
$s_t^{(1)}$
$s_t^{(2)}$
$s_t^{(3)}$
$s_t^{(4)}$
$s_t^{(5)}$
$s_t^{(6)}$
$s_t^{(7)}$
$s_t^{(8)}$
$s_t^{(9)}$
1.0
0.8
0.6
0.4
0.2
0.0
0
25
50
75
100
125
150
175
200


Actions
$a_t^{(1)}$
$a_t^{(2)}$
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
0
25
50
75
100
125
150
175
200


Reward
r_t
6
5
4
3
2
1
0
0
25
50
75
100
125
150
175
200

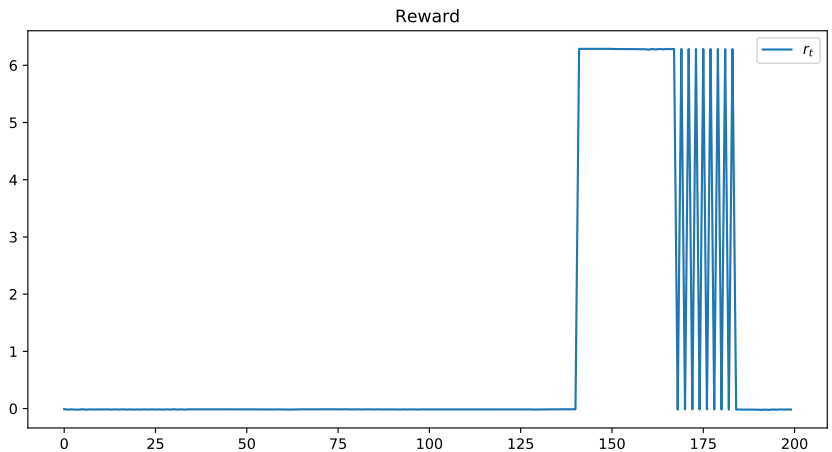
The Reward Signal

At time t we observe s_t, take action a_t, and obtain reward r_{t+1}. Over an episode, we see

$$s_1, a_1, \quad r_2, s_2, a_2, \quad r_3, s_3, a_3, \quad \ldots \quad , r_T, s_T$$

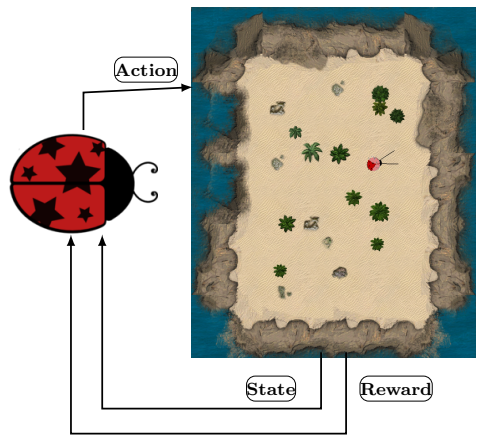
Problems:

- Reward signal is sparse: $r_t = 0$ or $r_t = -1$ most of the time
- . . . or weak: limited impact of best/worst actions on reward
- The temporal credit assignment problem: r_{t+1} may not be a result of a_t in any way; so which actions led to reward r_t?



Setting up the problem

- $\mathcal{S}$ state space, $s_t \in \mathcal{S}$
- $\mathcal{A}$ action space, $a_t \in \mathcal{A}$
- r reward function, $r_t = r(s_t, a_t)$
- p transition function, $s_{t+1} \sim p(\cdot|s_t, a_t)$



While s_t is not final:

1. $s_t, r_t \leftarrow \texttt{Env.step}(a_{t-1})$ ▷ Act upon environment; observe
2. $a_t \leftarrow \pi(s_t)$ ▷ Decide on an action
3. $t \leftarrow t + 1$ ▷ Continue

where $\pi : \mathcal{S} \mapsto \mathcal{A}$ is our policy.

The Policy

The policy defines the behaviour of the agent:

$$a_t = \pi(s_t)$$

indicates the action to take given the observation of the current state; i.e., the agent observes s_t and takes action a_t.

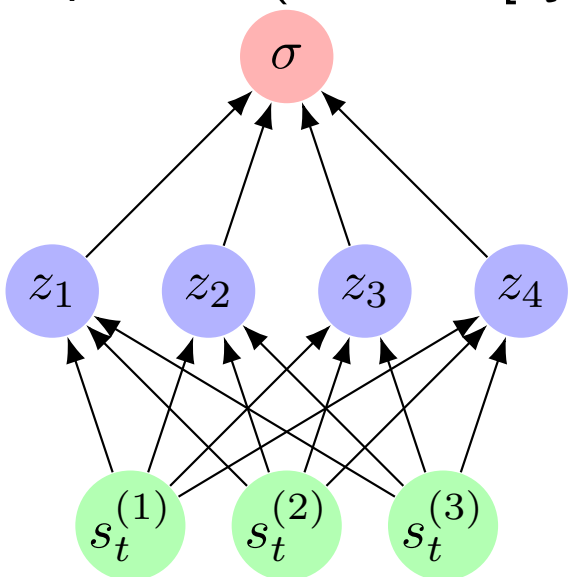
The Policy

The policy defines the behaviour of the agent:

$$a_t = \pi(s_t)$$

indicates the action to take given the observation of the current state; i.e., the agent observes s_t and takes action a_t.

Example policies (let $s_t = [s_t^{(1)}, s_t^{(2)}, s_t^{(3)}]$, $a_t \in \{-1, +1\}$):



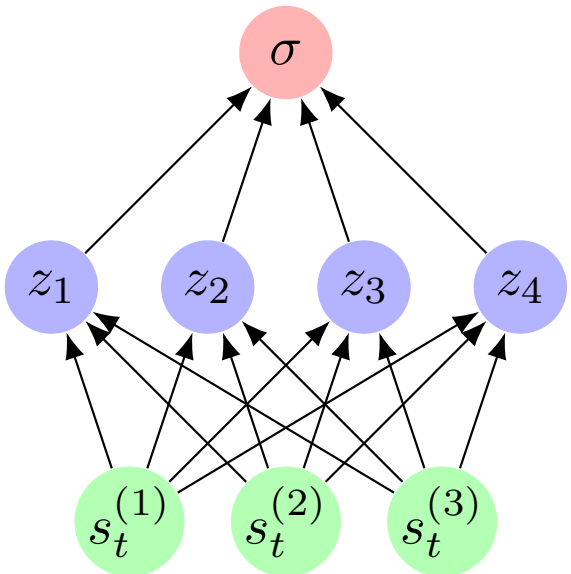
State s_1, s_2, s_3	Action a_t
0,1,0	-1
1,1,0	-1
0,1,1	+1
...	...

A policy can be any kind of mapping from states to action, e.g., via neural network, lookup (Q-)table, set of rules.

Stochastic Policy

A stochastic policy, where π provides a conditional probability distribution over actions given current state, and

$$a_t \sim \pi(\cdot|s_t) \quad \text{where } \pi(s_t) \in [0, 1]$$

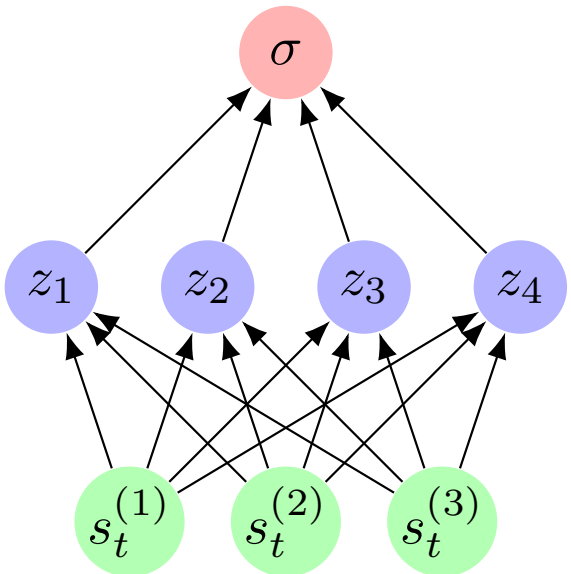


State s_1, s_2, s_3	$\pi(-1\|s_t)$ σ	$\pi(+1\|s_t)$ $1-\sigma$
0,1,0	0.1	0.9
1,1,0	0.5	0.5
0,1,1	1	0
...	...	...

Stochastic Policy

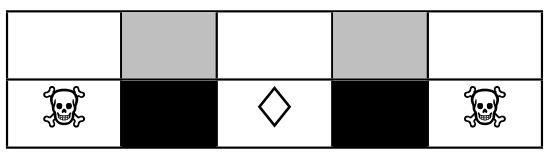
A stochastic policy, where π provides a conditional probability distribution over actions given current state, and

$$a_t \sim \pi(\cdot|s_t) \quad \text{where } \pi(s_t) \in [0, 1]$$



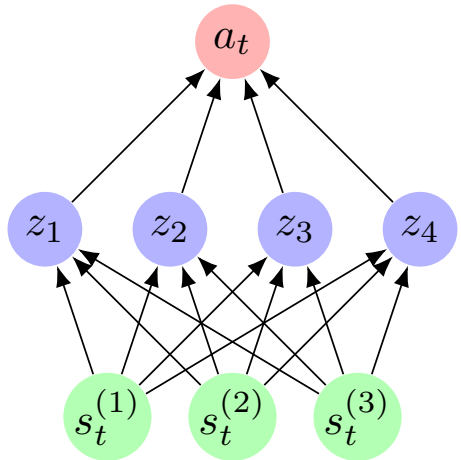
State s_1, s_2, s_3	$\pi(-1\|s_t)$ σ	$\pi(+1\|s_t)$ $1-\sigma$
0,1,0	0.1	0.9
1,1,0	0.5	0.5
0,1,1	1	0
...	...	...

- Built-in exploration
- In competitive games, easier to defeat deterministic agents
- Aliased states: States that appear the same, but are not



Policy Search

Direct Policy Search (Black Box Optimization)



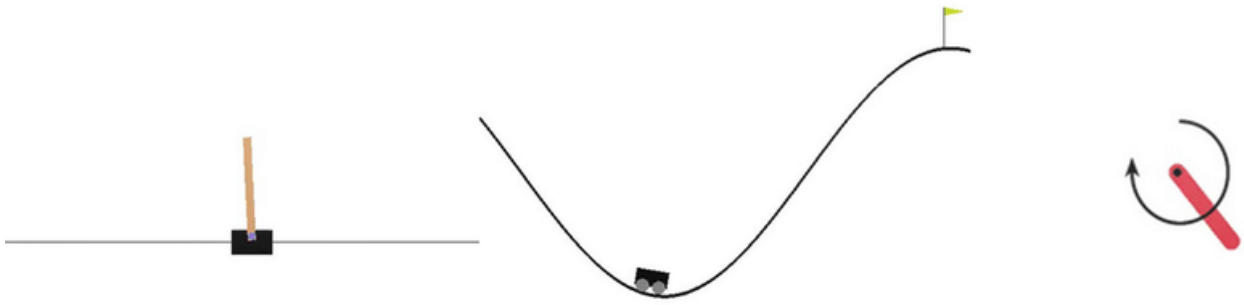
Suppose a parametric policy, parameters θ (e.g., of a neural network agent); i.e.,

$$a_t = \pi_\theta(s_t)$$

we can search directly in parameter space, i.e., policy search:

$$\theta^* = \underset{\theta \in \Theta}{\text{argmax}}\, g(\theta)$$

where $g(\theta)$ indicates how good π_θ is, e.g., $g(\theta) = 10 \Leftrightarrow$ agent survived 10 seconds / obtained score of 10 at end of episode.



- Many off-the-shelf search & optimization tools are available
- Works 'embarrassingly well' on some problems
- Handles continuous state/action spaces naturally
- No need to study reinforcement learning

But

- Expensive to evaluate $g(\theta)$ (run simulations/play games)
- Noisy (large $\mathbb{V}[g(\theta)]$); randomness from the environment!
- Curse of dimensionality: optimization in high dimensions (> 100) is very hard – it's why we don't train classifiers with black-box optimization!

Fully Observed Deterministic Environments

Fully Observed Deterministic Environments

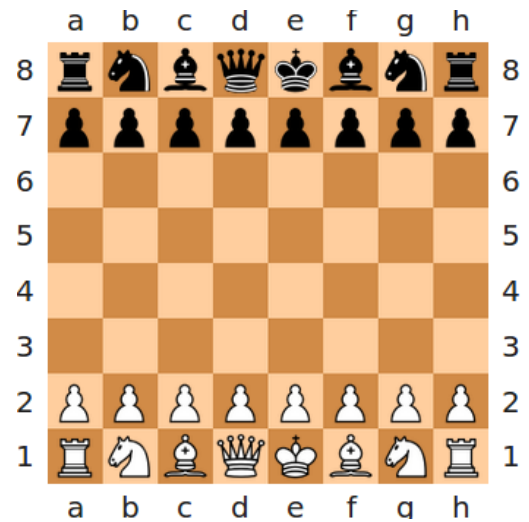
In deterministic environments: for any given action, the next state is known for certain.

In fully observed environments: s_t is a complete view[1] of the environment at time t.

For example, 8-puzzle, Noughts & crosses (tic tac toe), Chess, and many other board games.

1	3	5
7	2	4
8		6

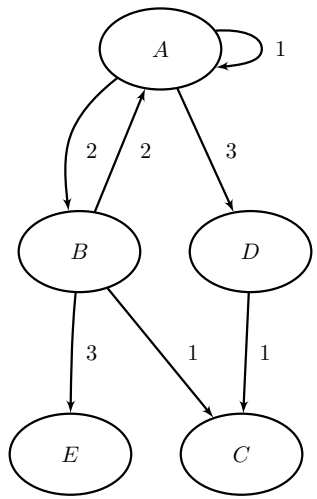
×		○
	○	×
×		○



[1] Otherwise it is a partially-observed environment

Fully Observed Deterministic Environments

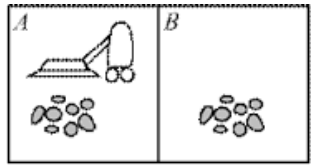
- $\mathcal{S}$ state space (e.g., $\mathcal{S} = \{A, B, C, D, E\}$), $s_t \in \mathcal{S}$
- $\mathcal{A}$ action space (e.g., $\mathcal{A} = \{1, 2, 3\}$), $a_t \in \mathcal{A}$
- r reward function, $r_t = r(s_{t-1}, a_t, s_t)$
  (e.g., $r(D, 1, C) = 1$, $r(B, 3, E) = -1$, and else $r(\cdot, \cdot, \cdot) = 0$); i.e., $r_t \in \{-1, 0, +1\}$



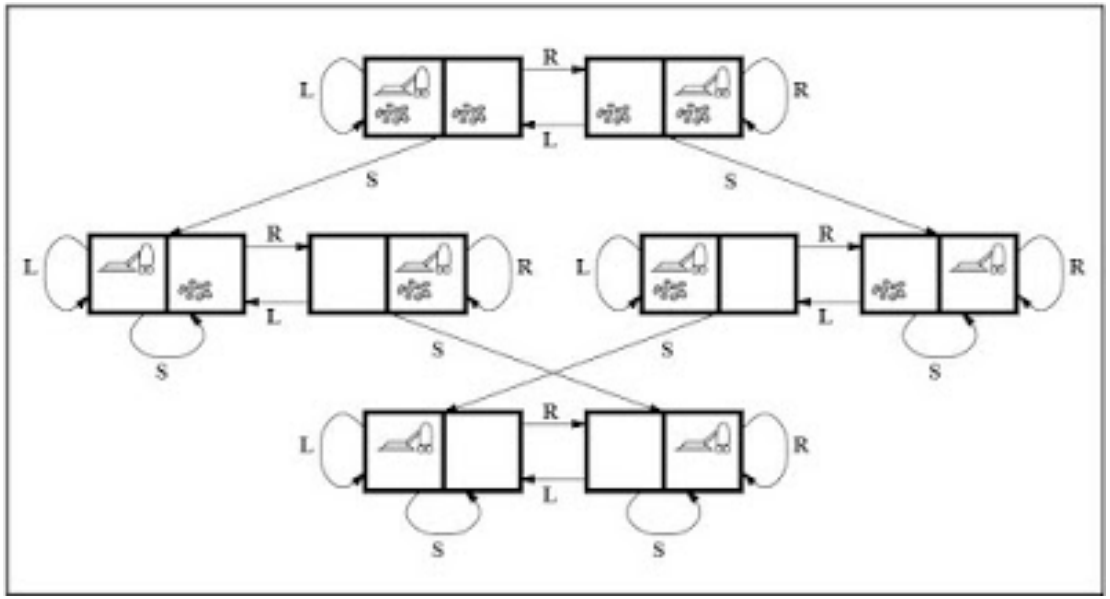
Actions shown on edges.

An epsiode/trace, e.g., $\{A, 0, 1, A, 0, 3, D, 0, 1, C, 1\}$.

For example, Vacuum Cleaner World:

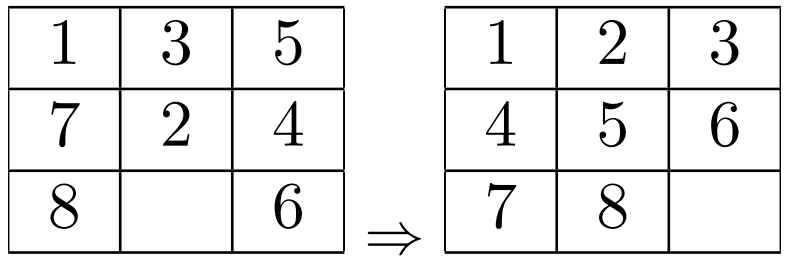


There are 8 possible states:



From Russel and Norvig, *Artificial Intelligence: A Modern Approach (2nd ed.)* 2002

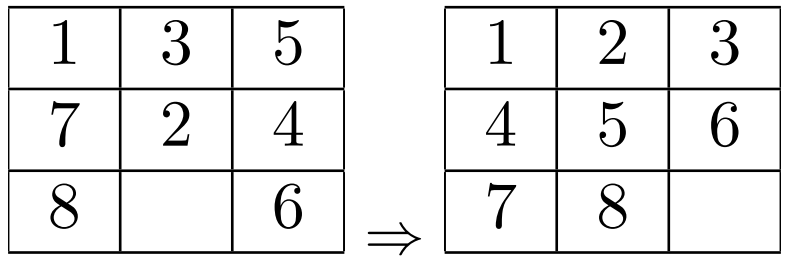
For example, the `8-puzzle`,



It's just a search (each node is a state):

1. Generate the search tree to goal state
2. Apply payoff to leaf
3. Backup
4. Choose the best branch

For example, the `8-puzzle`,



It's just a search (each node is a state):

1. Generate the search tree to goal state
2. Apply payoff to leaf
3. Backup
4. Choose the best branch

What about against an adversary (e.g., tic tac toe)? Can be made deterministic via minimax algorithm.

What if the branching factor is too large and/or long route to goal state (e.g., chess)? Simple search won't work.

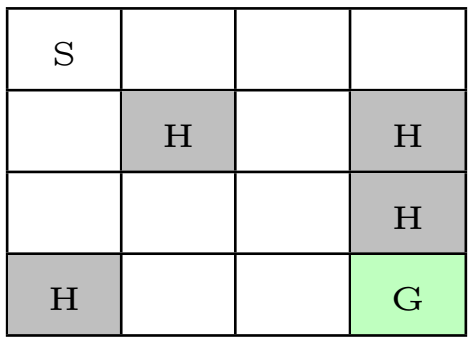
Stochastic Environments and MDPs

Stochastic Environments

In a stochastic environment, the agent takes action a from state s and arrives to state s' *with probability*[2]

$$s_{t+1} \sim p(s'|s, a)$$

The FROZEN LAKE environment: Get from Start state to Goal state without falling through a Hole in the ice. The ice is slippery, e.g., $p([1,1]|[2,1], \text{LEFT}) = 0.33$.



Not a search path from start to goal!

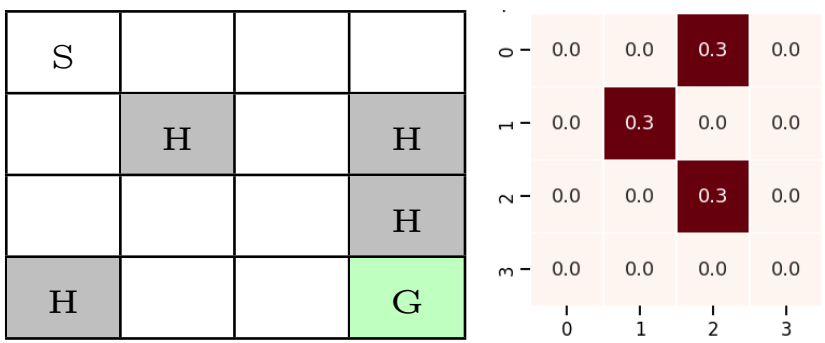
[2] N.B. $p(s'|s, a) \equiv \Pr[S_t = s'|S_t = s, A_t = a]$

Stochastic Environments

In a stochastic environment, the agent takes action a from state s and arrives to state s' *with probability*[2]

$$s_{t+1} \sim p(s'|s, a)$$

The FROZEN LAKE environment: Get from Start state to Goal state without falling through a Hole in the ice. The ice is slippery, e.g., $p([1,1]|[2,1], \text{LEFT}) = 0.33$.



Not a search path from start to goal!

[2] N.B. $p(s'|s, a) \equiv \Pr[S_t = s'|S_t = s, A_t = a]$

A Markov Decision Process (MDP)

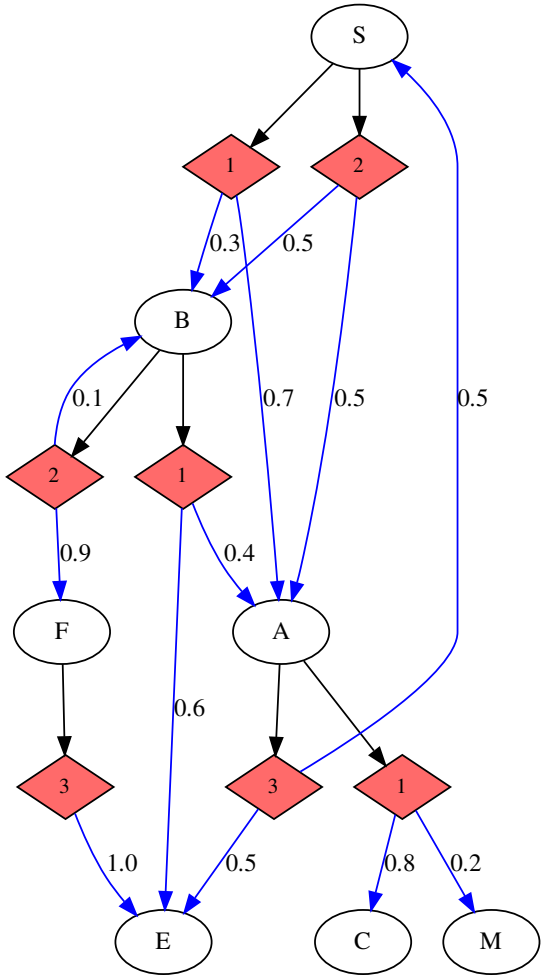
MDP: A model of the environment.

- $\mathcal{S}$ state space (e.g., {S,B,F,A,E,C,M})
- $\mathcal{A}$ action space (e.g., {1,2,3})
- $r(s, a)$ or $r(s, a, s')$ reward function
- $p(s'|s, a)$ transition function

▷ Compactly: $p(s', r \mid s, a)$

We do not necessarily know this model.

Goal: Learn policy π, which takes the best action from any given state.



Action in diamonds, p shown on edges; $r(\mathtt{A}, 1, \mathtt{M}) = 1$ and $r(s_{t-1}, a_t, s_t) = 0$ elsewhere

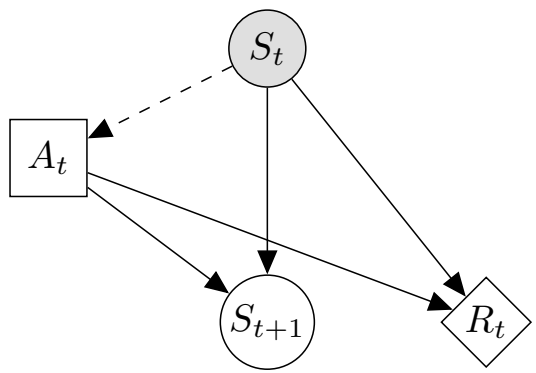
The Markov Property

MDPs provide a framework for modeling sequential decision making in stochastic environments. Most reinforcement learning problems can be framed as an MDP.

- **M** The Markov property
- **D** We add decisions (motivated by rewards)
- **P** A [Markov] process/chain[3]

Markov property: The effects of action a_t from state s_t depend only on those values;

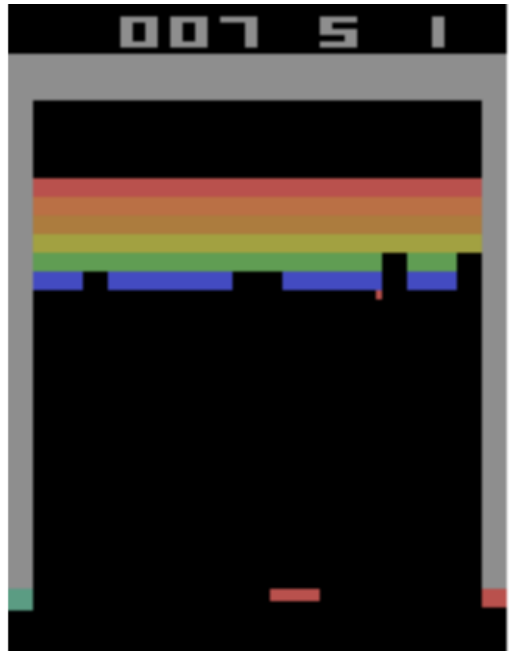
$$p(s_{t+1}|s_t, a_t) = p(s_{t+1}|s_1, \ldots, s_t, a_1, \ldots, a_t)$$



[3] Remark: If $P(s_{t+1}|s_t, a_t) = P(s_{t+1}|s_t)$ we reduce to a Markov chain

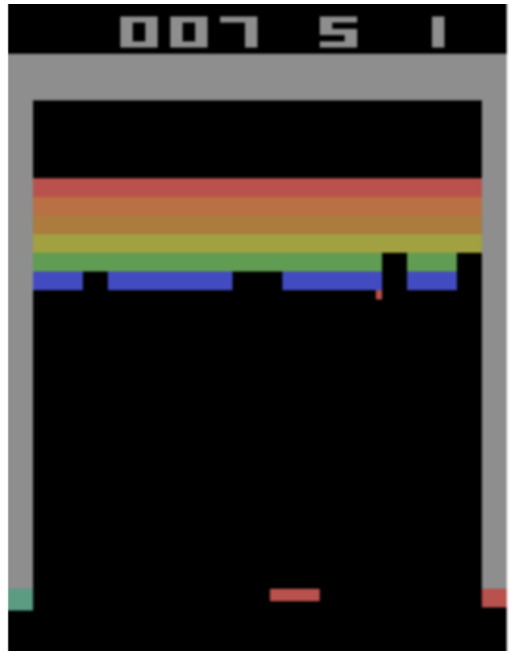
Does your environment satisfy the Markov property?

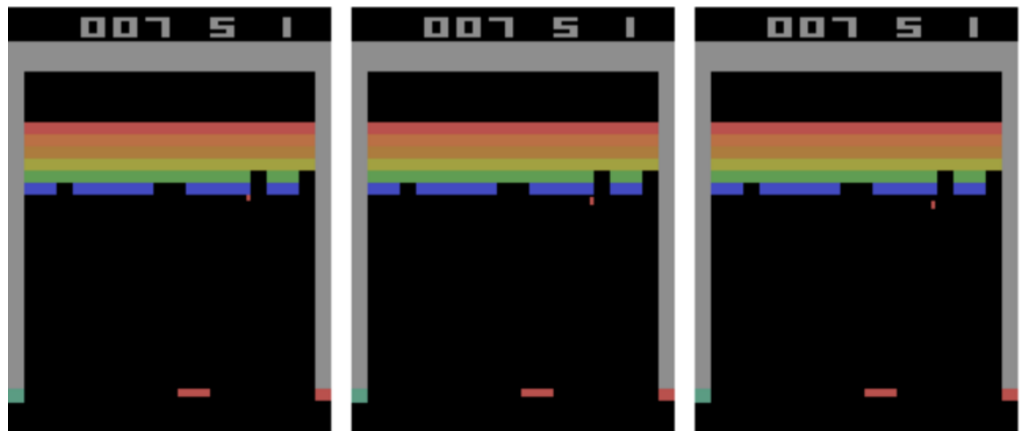
(Can you (an agent) perform optimally from state s_t?)

$s_t =$ 

Does your environment satisfy the Markov property?

(Can you (an agent) perform optimally from state s_t?)

$s_t =$ 

$s_t =$ 

Other solutions: LSTMs, track differences.

Policy, Return/Gain and the Bellman Equations

The Policy

A reminder:

$$a_t = \pi(s_t)$$

(should work for all $s_t \in \mathcal{S}$). But which policy is best?

The Policy

A reminder:

$$a_t = \pi(s_t)$$

(should work for all $s_t \in \mathcal{S}$). But which policy is best?

The best policy should take the best action from the current state;

$$a_t^* = \pi(s_t)$$

i.e., the action to to optimize . . .

The Policy

A reminder:

$$a_t = \pi(s_t)$$

(should work for all $s_t \in \mathcal{S}$). But which policy is best?

The best policy should take the best action from the current state;

$$a_t^* = \pi(s_t)$$

i.e., the action to to optimize . . .

major question: **what are we optimizing?**

. . . the reward at the next step (r_{t+1})? *No*, consider –

- what if the reward is only given at the end (r_T)
- what if a_t has nothing to do with r_{t+1}
- $r_{t+1} \sim R_{t+1}$ is a random variable; inherits randomness from the stochastic environment (and a stochastic policy)

The Return/Gain (Finite Scenario)

The return (aka gain) at step t is

$$\begin{aligned} G_t &= \sum_{i=t}^{T} R_i \\ &= R_t + R_{t+1} + R_{t+2} + \ldots + R_T \end{aligned}$$

i.e., the sum of rewards until the end of the episode; it indicates the value at current time t.

The Return/Gain (Finite Scenario)

The return (aka gain) at step t is

$$\begin{aligned} G_t &= \sum_{i=t}^{T} R_i \\ &= R_t + R_{t+1} + R_{t+2} + \ldots + R_T \end{aligned}$$

i.e., the sum of rewards until the end of the episode; it indicates the value at current time t.

Considerations:

- What is T? What if $T = \infty$?
- Does it make sense that G_t is the same when $r_{t+1} = 1$ vs when $r_{t+1000} = 1$ (and all others 0)?

The Return/Gain (Infinite Scenario)

The return (aka gain) at step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1} \tag{1}$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

for discount factor $\gamma \in (0, 1)$ which indicates the relative value of closer rewards.

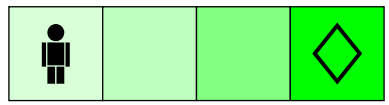
The Return/Gain (Infinite Scenario)

The return (aka gain) at step t is

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{k+t+1} \tag{1}$$

$$= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \tag{2}$$

for discount factor $\gamma \in (0, 1)$ which indicates the relative value of closer rewards.



Task: The agent should take actions A_t to maximize G_t!

Problem: G_t is from the future (may be even $t = \infty$)!
G_t inherits the randomness from all $R_{t+1}, R_{t+2}, \dots$

Solution: We should maximize expected return $\mathbb{E}[G_t]$.

The Value Function

The value function (aka state-value function),

$$V^{\pi}(s) = \mathbb{E}[G_t | S_t = s]$$

maps a state to a value.

The value of a state s is the expected return from that state following policy π.

We may think of a vector of $|\mathcal{S}|$ values; if $\mathcal{S} = \{s_1, s_2, s_3, s_4\}$:

	s_1	s_2	s_3	s_4
V				

The policy is implicit: move to states with high value.

A V-function and Policy for the Frozen Lake Environment:

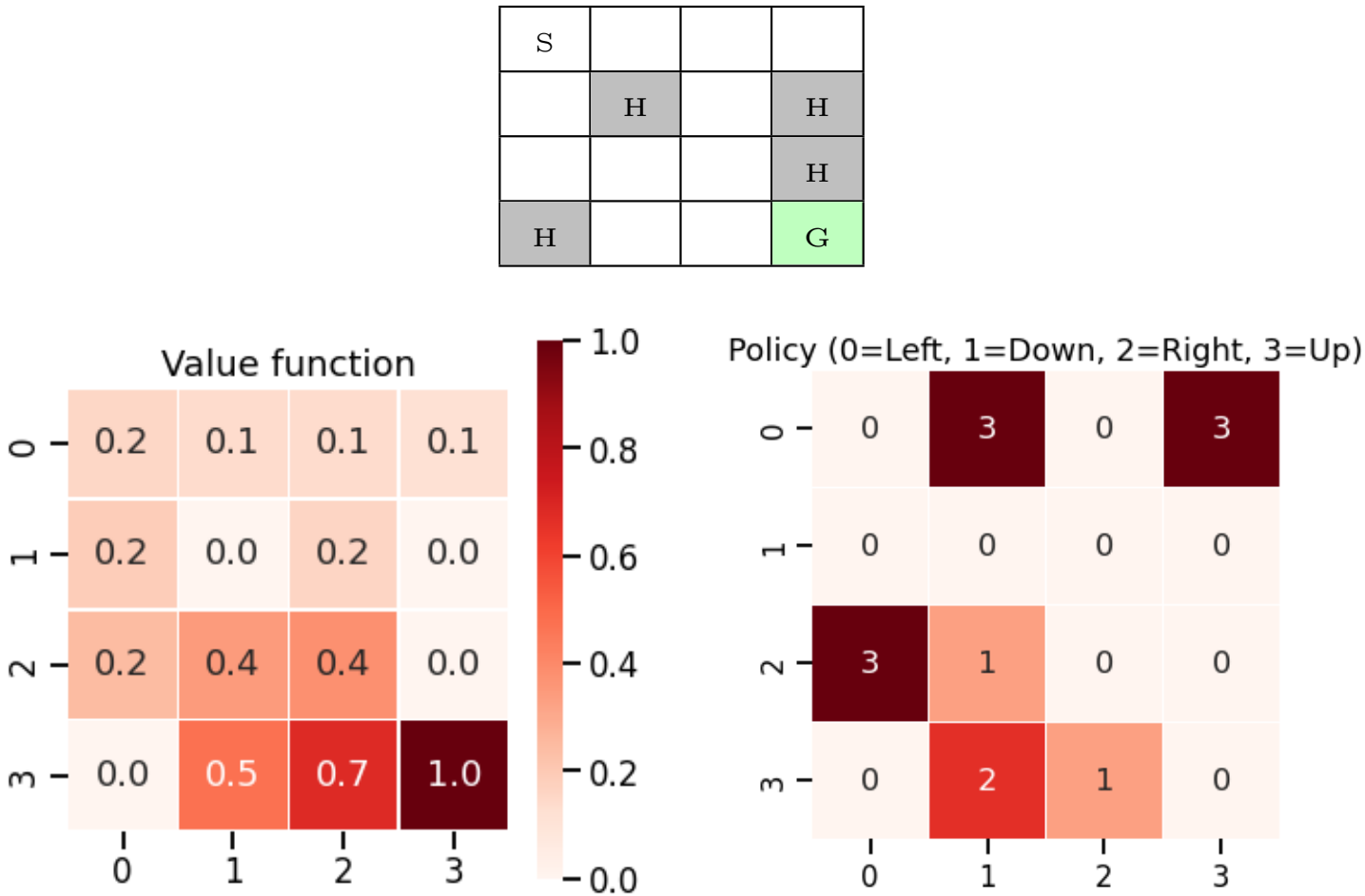


Image credits: Jérémie DeCock (TD teacher)

The Action-Value Function

The action-value function,

$$Q^{\pi}(s, a) = \mathbb{E}[G_t | S_t = s, A_t = a]$$

maps a state *and action* to a value.

We may think of a table of $|\mathcal{S}| \times |\mathcal{A}|$ values; with $\mathcal{A} = \{a_1, a_2\}$:

Q	a_1	a_2
s_1		
s_2		
s_3		
s_4		

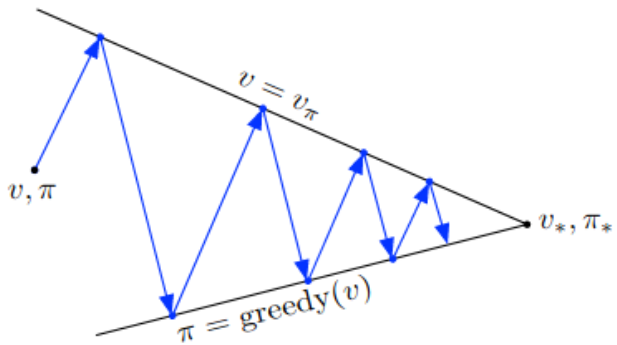
The greedy policy is explicit:

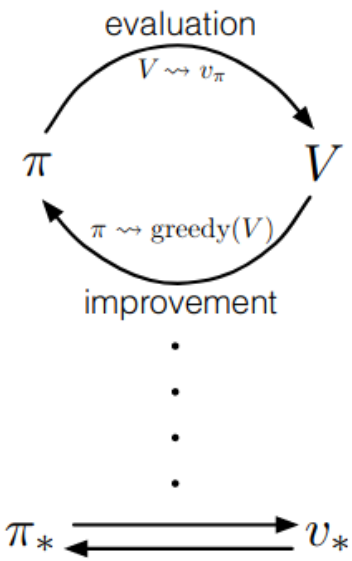
$$a_t = \pi(s_t) = \underset{a \in \mathcal{A}}{\text{argmax}}\, Q(s_t, a)$$

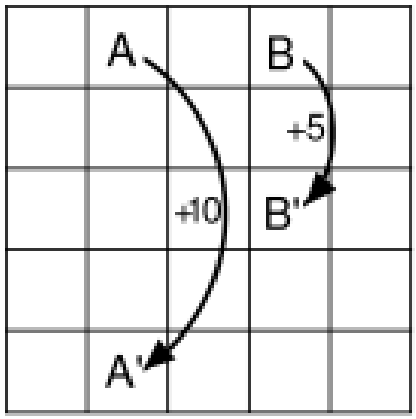
Bellman Optimality Equation

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi_*}(s, a) \\
&= \ldots \quad \triangleright \text{ See } \textit{Sutton and Barto, 2018} \text{ for derivation} \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma V^*(s') \right] \\
\pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}(s)} Q^{\pi_*}(s, a)
\end{aligned}
$$

Note the max and the argmax; relationship between π^* and V^*.



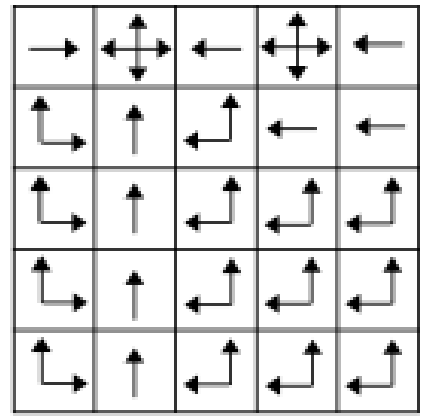




a) gridworld

22.0	24.4	22.0	19.4	17.5
19.8	22.0	19.8	17.8	16.0
17.8	19.8	17.8	16.0	14.4
16.0	17.8	16.0	14.4	13.0
14.4	16.0	14.4	13.0	11.7

b) V^*



c) π^*

Ref: Sutton and Barto. *Reinforcement Learning: An Introduction*, 2018.

In reinforcement learning, we do not know the underlying system dynamics of the MDP, we need to learn it through interaction with the environment.

Bellman Equation for V^π (Preliminaries)

Recall (Slide 31) the value function:

$$V^\pi(s) = \mathbb{E}[G_t | S_t = s] \tag{3}$$

A recursion trick we will need:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots) \\ &= R_{t+1} + \gamma G_{t+1} \quad \triangleright \text{ Recursion!} \end{aligned} \tag{4}$$

A reminder on conditiontal probability and expectations and marginalizing out (some function g):

$$\mathbb{E}_{A,R,S'}[g(A, R, S') | S = s] = \sum_{a,r,s'} g(a, r, s') P(a, r, s' | s)$$

Total law of expectation:

$$\mathbb{E}[G] = \mathbb{E}_S[\mathbb{E}[G|S]]$$

Bellman Equation V^π (Derivation)

From an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \quad \triangleright \text{ Cond. expectation; from Eq. (3)} \qquad (5) \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \middle| S_t = s \right] \quad \triangleright \text{ Recursion; from Eq. (4)} \\
&= \sum_{a,s',r} \underbrace{p(a|s)p(r|s,a)p(s'|s,a)}_{p(a,r,s'|s)} \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s' \right] \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s' \right] \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma V_\pi(s') \right] \quad \forall s \in \mathcal{S} \qquad (6)
\end{aligned}
$$

Where the final line completes the recursion via Eq. (5).

Bellman Equation V^π (Derivation)

From an MDP $(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma)$:

$$
\begin{aligned}
V^\pi(s) &= \mathbb{E}_\pi \left[G_t | S_t = s \right] \quad \triangleright \text{ Cond. expectation; from Eq. (3)} \qquad (5) \\
&= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1} \middle| S_t = s \right] \quad \triangleright \text{ Recursion; from Eq. (4)} \\
&= \sum_{a,s',r} \underbrace{p(a|s)p(r|s,a)p(s'|s,a)}_{p(a,r,s'|s)} \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s' \right] \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma \mathbb{E}_\pi \left[G_{t+1} | S_{t+1} = s' \right] \right] \\
&= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma V_\pi(s') \right] \quad \forall s \in \mathcal{S} \qquad (6)
\end{aligned}
$$

Where the final line completes the recursion via Eq. (5).

Why is this important? It's the MDP + recursion! (the 'solution'!)
We just need V^* ...

Bellman Optimality Equation V^*

The optimal policy is

$$\pi^* = \text{argmax}_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

The value function associated with this policy is:

$$V^*(s) = \text{max}_{\pi} V^{\pi}(s) \quad \forall s \in \mathcal{S}$$

Note that (the value from state s, under policy π)

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s))$$

It means that (from Eq. (6)):

$$Q^{\pi}(s, \pi(s)) = \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(r, s'|s, a) \left[r + \gamma V_{\pi}(s') \right] \qquad \forall s \in \mathcal{S}$$

The action-value function Q shares the same optimal policy.

Bellman Optimality Equation

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi_*}(s, a) \\
&= \dots \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma V^*(s') \right] \\
\pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}(s)} Q^{\pi_*}(s, a)
\end{aligned}
$$

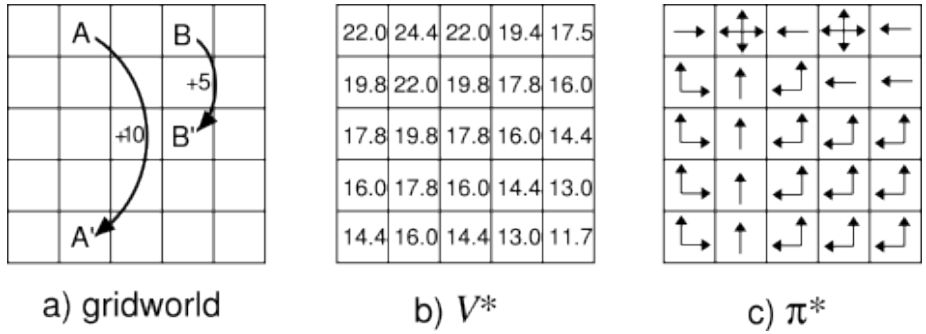
(again the max and the argmax – back to Slide 38).

Bellman Optimality Equation

$$
\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}(s)} Q^{\pi_*}(s,a) \\
&= \dots \\
&= \max_{a \in \mathcal{A}(s)} \sum_{s' \in \mathcal{S}, r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma V^*(s') \right] \\
\pi^*(s) &= \operatorname*{argmax}_{a \in \mathcal{A}(s)} Q^{\pi_*}(s,a)
\end{aligned}
$$

(again the max and the argmax – back to Slide 38).

So, initial and final question: how to get π^*?

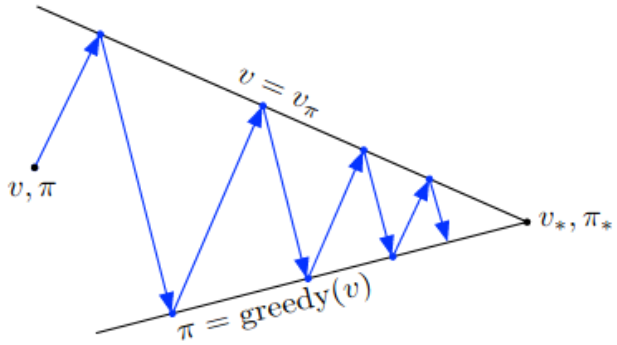


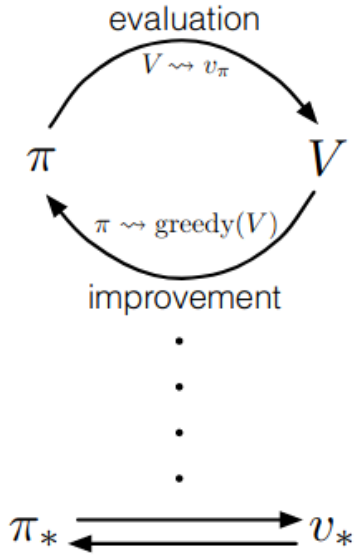
Ref: Sutton and Barto. *Reinforcement Learning: An Introduction*, 2018.

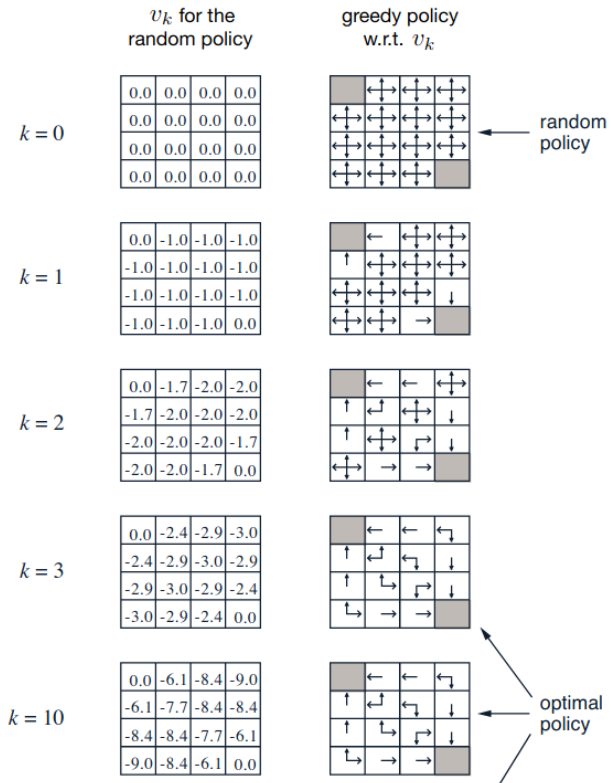
Iterative solutions:

- Value iteration
- Policy iteration
- Q-learning
- Sarsa
- ...

In reinforcement learning, we do not know the underlying system dynamics of the MDP, we need to learn it through interaction with the environment (but we can learn it).


$v = v_\pi$
v, π
v_*, π_*
$\pi = \text{greedy}(v)$


evaluation
$V \rightsquigarrow v_\pi$
π
V
$\pi \rightsquigarrow \text{greedy}(V)$
improvement
π_*
v_*


v_k for the random policy
greedy policy w.r.t. v_k
$k = 0$
$k = 1$
$k = 2$
$k = 3$
$k = 10$
random policy
optimal policy

Exploration vs Exploitation

We need to learn the system dynamics through interaction.

One possibility: Use a behaviour policy (possibly random) μ:

$$a_t \sim \mu$$

(this is an off-policy method).

1. Play many episodes with policy μ; record G_t for each (s_t, a_t)
2. Use these samples to approximate the expectation:

$$Q^{\mu}(s, a) = \frac{1}{n} \sum_{i=1}^{n} G_t^{(i)} \approx \mathbb{E}[G_t | s, a]$$

3. Employ the greedy policy π on Q.

Trade-off: Exploration vs exploitation.

This in Monte Carlo RL. More on Monte Carlo next time!

Types of Reinforcement Learning Agents

Value-Based or Policy-Based ?

- Value Based (value function; policy is implicit)
- Policy Based (no value function)
- Actor Critic (both value function + policy)

Model Free vs Model Based: We can build a model of the environment, or do without it.

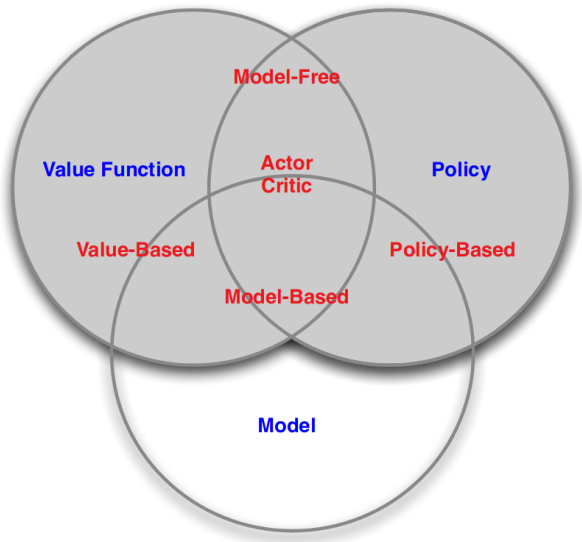


Image from [1]

Summary / Important Concepts

- Agent and Environment
- States, Actions, Reward
- Model of the environment (MDP)
  - Markov property
  - Deterministic vs stochastic
  - Unknown vs fully observable vs partially observable
- Return/Gain (finite horizon vs infinite horizon; γ)
- Policy (deterministic policy, stochastic policy, greedy policy)
- State Value and Action-Value functions
- Bellman Equations
- Exploration vs Exploitation trade-off

Reinforcement learning is difficult (compared to supervised learning), but there are many applications where supervised learning does not apply.

Deep Reinforcement Learning

[2]

Game mechanics still relatively simply (move, left, right, shoot, . . .), but state space is huge! Q-table impossible!

A Deep Representation of the Q-table

We replace the Q table with a neural network:

$$Q(s, a) = q_{\boldsymbol{w}}(s)_a$$

where $q_{\boldsymbol{w}}$ is a (deep) neural network parametrized by $\boldsymbol{w}$.

For example,

$$[Q(s, a_1), \ldots, Q(s, a_4)] = q_{\boldsymbol{w}}(s)$$

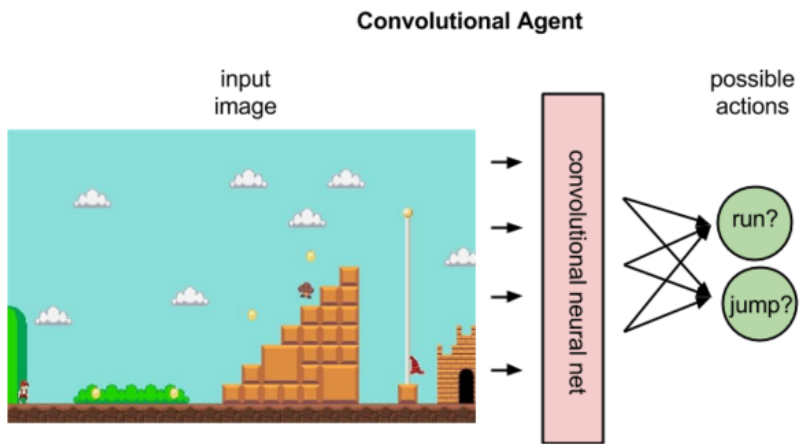
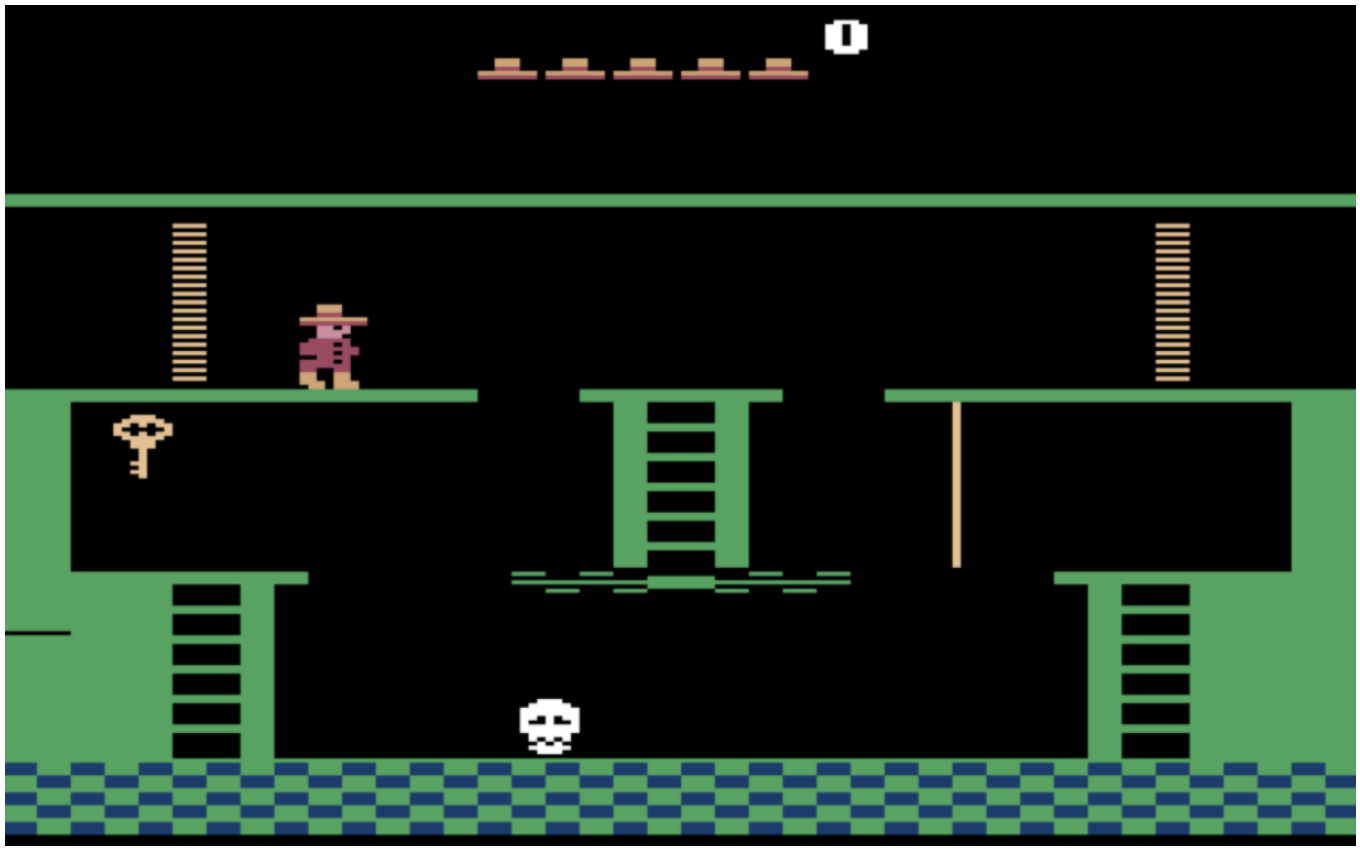


Image Source: [3]

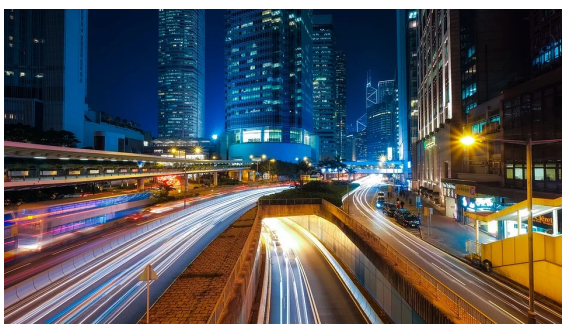
Montezuma's Revenge (much harder than FPS!)



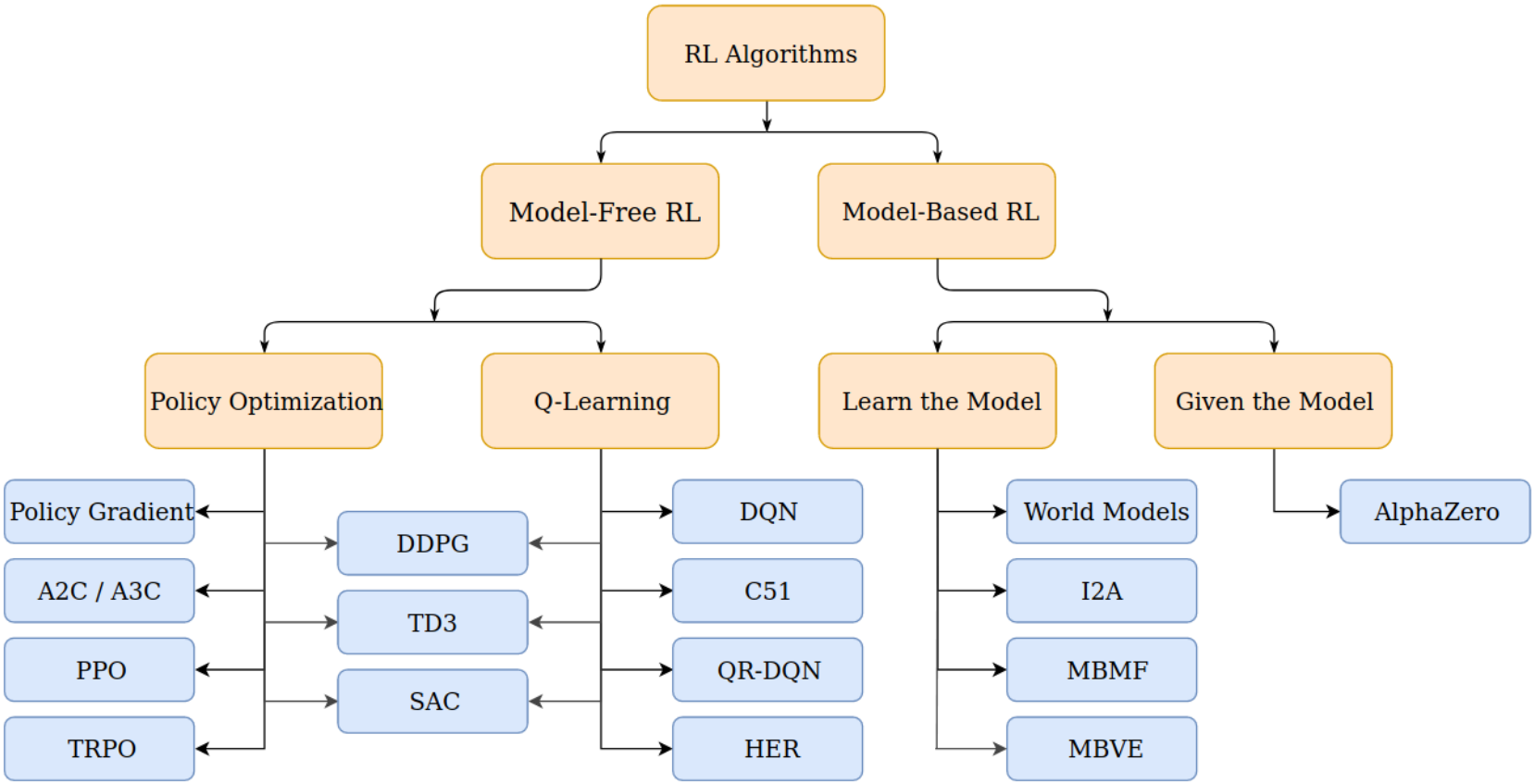
Montezuma's Revenge – Notoriously Difficult for RL Agents [4]

Modern Challenges

- Learning quickly with limited samples
- Safety issues
- Explainable agents, interpretable actions
- Multi-objective reward fns, multi-agent
- Real-time learning and inference
- Delayed/partial/weak observations



A Taxonomy of Methods



Modern Machine Learning Methods [5]

Reinforcement Learning I

INF581 Advanced Machine Learning and Autonomous Agents

Jesse Read